MSRC-5000SRD
Appendix E
rev. 1.2

Support and Rationale Document

for the

Software Communications Architecture Specification

APPENDIX E. API SUPPLEMENT

21 December 2000

Revision Summary

1.2	Original release

Table of Contents

List of Illustrations

1 INTRODUCTION

This appendix to the Software Communications Architecture (SCA) Support and Rationale Document (SRD) is a companion to the API Supplement to the SCAS and provides background and support information for the development of Joint Tactical Radio System (JTRS) software configurable radios. For easier tracking of requirements and rationale, the outline of this appendix is similar to the API Supplement in Sections 3 and 4. Together, the API Supplement and this SRD appendix provide the framework for communications between JTRS radio components for differing domains/platforms and selected waveforms. Neither document specifies implementation details for any particular domain or waveform application.

1.1 SCOPE

This document provides:

1. Rationale explaining technical and methodology decisions made in the development of the API Supplement.
2. Background explanation of the requirements in the API Supplement.

2 REFERENCE DOCUMENTS

The reference documents listed in section 2 of the API Supplement are referenced in sections 3 and 4 of the supplement.

3 APPLICATION PROGRAMMING INTERFACES

3.1 GENERAL

A key component of the cost reduction strategy for radio procurement embodied in JTRS is the ability to procure portable waveforms (portable at the source code level, not the binary level) which may run on multiple vendors' platforms. In addition, maintenance and upgrade of individual components of a waveform without affecting other components requires that the components have a well defined interface. It is for this reason that standard interfaces are defined for JTRS.

3.2 JTRS APIS

The approach taken to define standardized APIs for JTRS is based of the original partitioning approach taken in Step 1 of the JTRS program that is illustrated in figure 3-1. Note that the partitioning of application elements is essentially based on the OSI model.

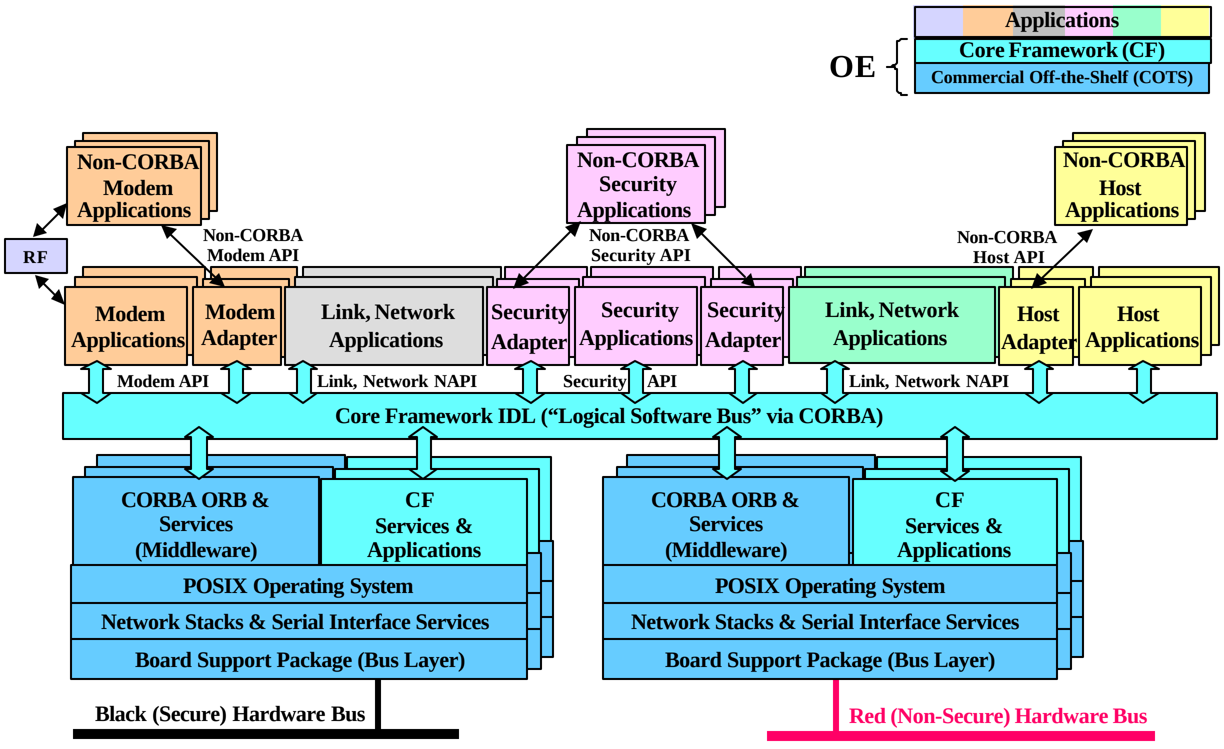
OE
Applications
Core Framework (CF)
Commercial Off-the-Shelf (COTS)
Non-CORBA Modem Applications
Non-CORBA Security Applications
Non-CORBA Host Applications
RF
Non-CORBA Modem API
Non-CORBA Security API
Non-CORBA Host API
Modem Applications
Modem Adapter
Link, Network Applications
Security Adapter
Security Applications
Security Adapter
Link, Network Applications
Host Adapter
Host Applications
Modem API
Link, Network NAPI
Security API
Link, Network NAPI
Core Framework IDL ("Logical Software Bus" via CORBA)
CORBA ORB & Services (Middleware)
CF Services & Applications
POSIX Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
CORBA ORB & Services (Middleware)
CF Services & Applications
POSIX Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black (Secure) Hardware Bus
Red (Non-Secure) Hardware Bus

Figure 3-1. Step 1 JTRS Architecture

Although the original architecture provided a preliminary partitioning of components, the interfaces defined were deemed insufficient in and of themselves to define a common set of interfaces for a given waveform. In addition, in the original partitioning, the definition of the modem component was too nebulous and lent itself to too wide an interpretation of its functionality. To rectify these problems a slightly more granular partitioning is defined with the additional requirement that standardized APIs must exist at the boundaries between components.

The new partitioning is shown in figure 3-2. Note that the Modem and Link components have been broken up into Physical, Media Access Control (MAC) and Logical Link Control components.

The network component is really a sub-network component with an LLC interface and not an inter-network (IP) component for which no standard interface has been defined. It is expected that commercial off-the-shelf (COTS) IP routers will often provide inter-networking functionality. The Transport Layer and Network Layer for these COTS components are usually closely coupled with a private interface. This is the main reason that an inter-networking interface is not specified.

The Host Application Component has been renamed to I/O.

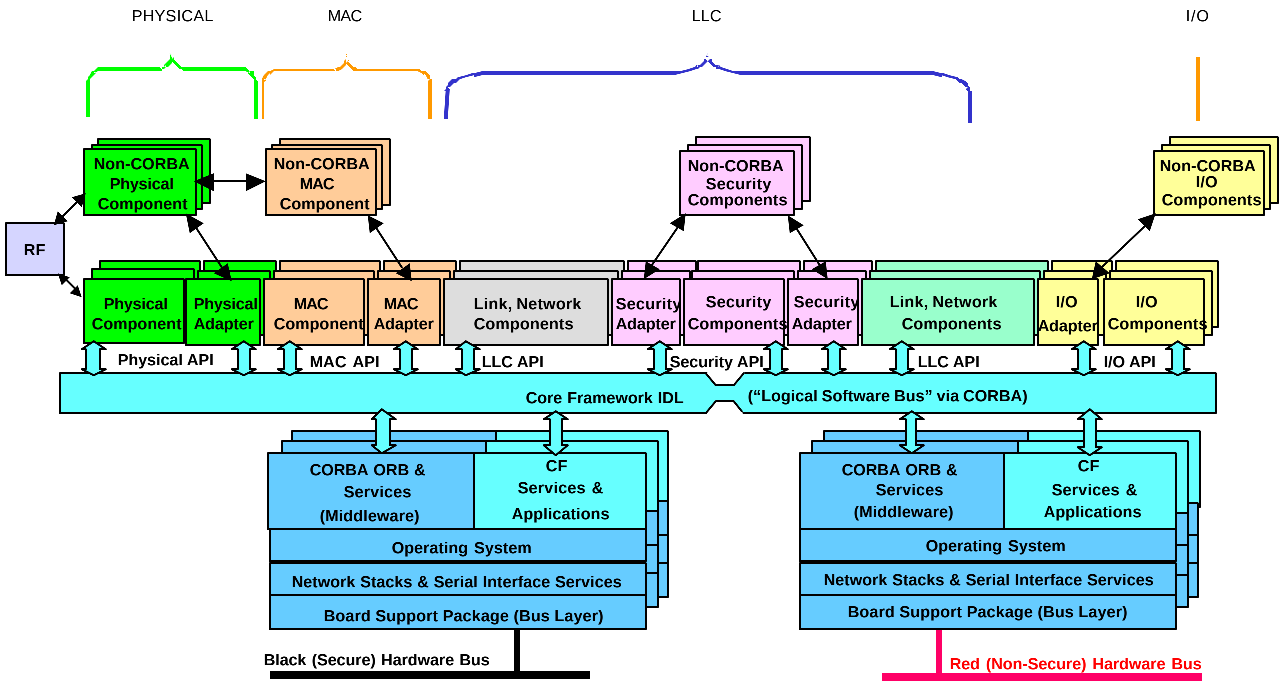
PHYSICAL
MAC
LLC
I/O
Non-CORBA Physical Component
Non-CORBA MAC Component
Non-CORBA Security Components
Non-CORBA I/O Components
RF
Physical Component
Physical Adapter
MAC Component
MAC Adapter
Link, Network Components
Security Adapter
Security Components
Security Adapter
Link, Network Components
I/O Adapter
I/O Components
Physical API
MAC API
LLC API
Security API
LLC API
I/O API
Core Framework IDL
("Logical Software Bus" via CORBA)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black (Secure) Hardware Bus
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red (Non-Secure) Hardware Bus

Figure 3-2. JTRS Component Partitioning and Associated APIs

The following list gives a brief explanation of each component category.

A. I/O. This is a domain component containing voice and/or data processing. Common audio and data interfaces are thus provided to any application needing such an interface.

B. Security. The interfaces to security applications, components, and devices are covered in the Security Supplement to the SCA.

C. Network. This is a sub-network component with an LLC API. An implicit Sub-Network Dependent Convergence Function (SNDCF) exists between the LLC API and the sub-network functionality. An inter-networking API is not defined.

D. Logical Link Control. This interface is at the component(s) used by waveform applications requiring link layer behavior (Data Link service conforming to the Open Systems Interconnection (OSI) model for networking systems).

E. MAC. This interface, while analogous to and fully supporting the services of the Medium Access Control sublayer of the OSI model Link Layer, is provided for waveform applications that have media access control behavior (e.g. transmit/receive time slot control in TDMA, error correction coding control, etc.).

F. Physical. This interface provides for translation from bits/symbols to RF and RF to bits/symbols for waveform transmission and reception.

Figure 3-2 shows these component types mapped to the software structure of the SCA.

Depending on the waveform, link behavior can be on either (or both) the Red or Black side. This partitioning is similar to the partitioning defined in the paper, "The Impact of Software Radio on Wireless Networking"[1]. The approach taken by the MSRC is less granular and the distribution of the functionality between OSI layers is a bit different (e.g. where spectral multiple access resides). The MSRC recognizes that in certain waveforms, functionality may exist in the Physical layer where for other waveforms it may exist in the MAC.

3.2.1 API Definition

Given the partitioning described in paragraph 3.2, rules and guidelines were established for defining APIs.

Figure 3-3 illustrates the first rule, which is to segregate the interfaces that support real time data flow and control (A) and the interfaces that support non-real time control, setup and initialization (B). These interfaces are logically separate because the former defines interfaces that are used in-line from I/O device to antenna while the latter define interfaces that are not logically tied to any in-line processing. Segregating these interfaces allows them to be optimized for their role in the architecture or the implementation of a waveform. To meet performance requirements for a particular waveform on a particular platform, two layers (e.g. MAC and Physical) may need to be tightly coupled and the A interface may not be implemented while the B interfaces remain unaffected. Again this approach is similar to the one taken in the V.G. Bose paper.

[1] V. G. Bose, "The Impact of Software Radio on Wireless Networking," Mobile Computing and Communications Review, Volume 3, No. 1, January 1999

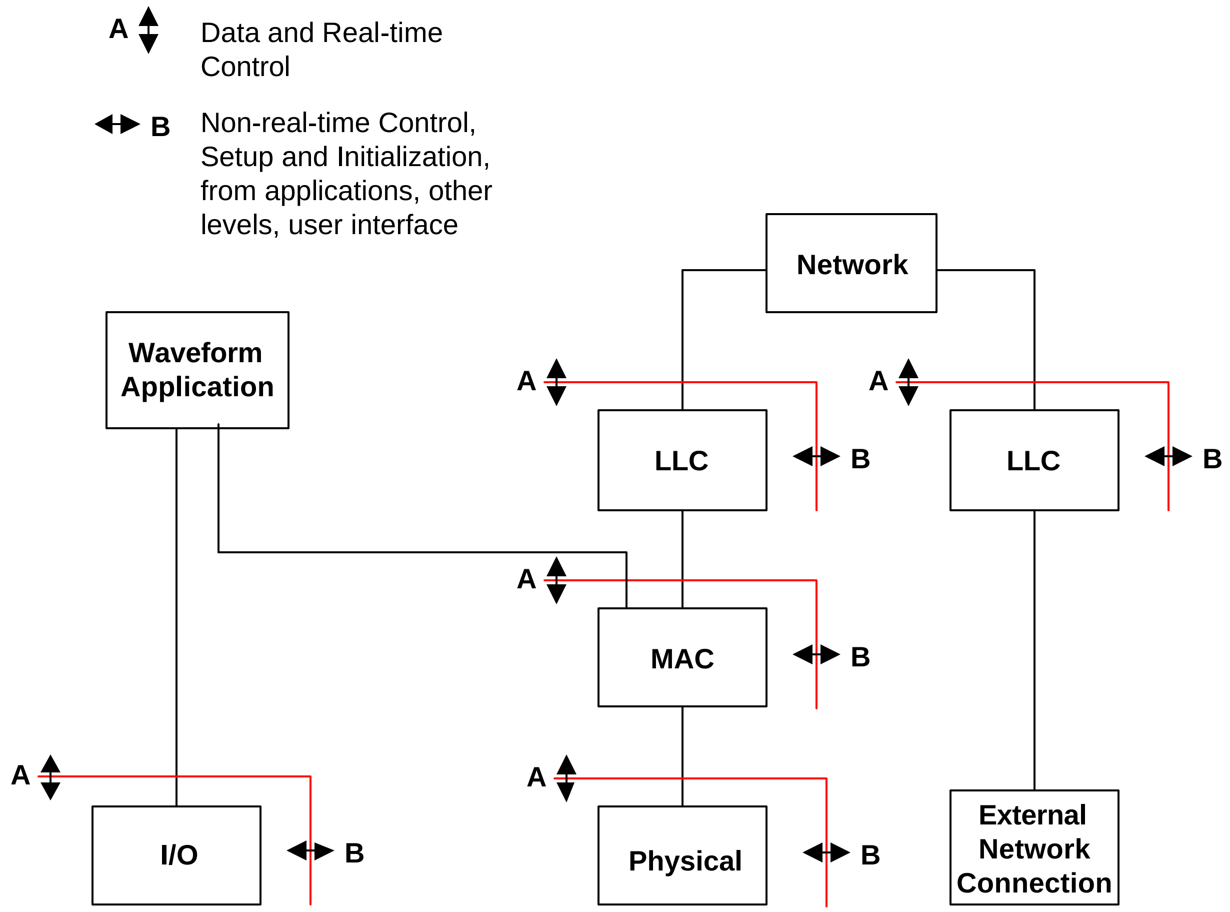
A
Data and Real-time Control
B
Non-real-time Control, Setup and Initialization, from applications, other levels, user interface
Network
Waveform Application
A
A
LLC
B
LLC
B
A
MAC
B
A
A
I/O
B
Physical
B
External Network Connection

Figure 3-3. Relationship Example of APIs

3.2.2 Relationship of APIs to the SCA

3.2.2.1 APIs, Components, and Ports

For a given waveform application the separate components of a waveform must be connected together in a manner consistent with the SCA. The interfaces defined for a given waveform must be allocated to components in the Domain Profile and the dependencies of components on these interfaces must be enumerated as well. The rationale for the Software Component and Software Assembly Descriptors is in the SCA SRD.

3.3 API Services

In order to promote interchangeability and portability of waveform components it is necessary to define APIs for the components. As long as a component obeys a well-defined API it can be interchanged with another component which obeys the same API.

3.4 BUILDING BLOCKS

The functionality encompassed in the Physical and MAC layers for differing waveforms, although similar in a general sense, can differ in the details. Control and status parameters, real time or non-real time, which apply to one waveform may be different or absent altogether for another waveform. This is especially true for the MAC layer as defined for JTRS. Given these differences, the question is how are the interfaces defined? Five alternatives were examined and rated by the MSRC Network Working Group according to criteria and associated figures of merit. The MSRC Network Working Group derived the criteria and figures of merit from the key performance parameters (KPP), threshold requirements and objectives stated in the ORD. The documentation of this analysis is in Appendix D of the SCA SRD. The result of this analysis was the building block approach (known as the commercial model with inheritance in Appendix D of the SCA SRD) where APIs can be built from smaller pieces.

4 REQUIREMENTS

4.1 GENERAL REQUIREMENTS

4.1.1 Name Scoping

There are no requirements levied in this section. The following is provided for clarification:

CORBA IDL provides the capability to provide scope to names of related entities through the module construct. It is urged that Building Block and API Service Definitions use the module construct to avoid name space pollution and to provide a built-in, coherent, consistent and easy to use naming convention. Once the naming convention for a waveform is established, most of the rest of the naming for the waveform falls into place. For example if the module name for Have Quick is HQ then the IDL would be:

```
module HQ {
        .
        .
        module Physical {
        .
        .
        }
        module MAC {
        .
        .
        }
}
```

In many cases within each module the names of the instantiated building blocks can remain unchanged because of the scope afforded by the module construct. An instantiation of the MediaSetup building block in the Physical layer would be referred to as HQ::Physical::MediaSetup using the above example.

4.2 API REQUIREMENTS

All SCA-compliant APIs shall have their interfaces described in IDL.

It is required that all service definitions interfaces be defined using Interface Description Language (IDL). If it is expected that many different vendors will reuse published APIs, it becomes important to have a standard interface description language. IDL was chosen because it provides a description that is independent of the programming language, it can be directly compiled into CORBA ORBs, and it fosters inheritance.

4.2.1 API Usage and Creation

The structure and language requirements of the APIs have been selected to provide commonality between implementations to foster reuse and portability of applications. To further these ends, one of the following methods for creating APIs shall be used.

- **A. Use existing API.**
- **B. Create a new API by inheriting an existing API and then extending its services.**
- **C. Translate an existing non-JTRS API to IDL to create a new JTRS API.**
- **D. Develop a new API based upon one or more Building Blocks. Use of Building Blocks should follow the order of using existing Building Blocks, extending existing Building Blocks, generating new Building Blocks.**

For these identified methods, the order of preference shall flow from Item A to Item D.

The steps provided in this section are given in the order of precedence to promote common interfaces between protocol layers. The most advantageous method is to reuse an interface verbatim. The second most advantageous method is to inherit pieces of existing interfaces at the same layer to promote common functions among protocol entities. Steps C and D require starting a new inheritance tree which is not as advantageous as steps A and B, but is required to incorporate future waveforms that are very different from existing waveforms and import commercial Service Definitions.

4.2.2 API Transfer Mechanisms

4.2.2.1 Standard Transfer Mechanism

The standard transfer mechanism shall be CORBA except as allowed in 4.2.2.2

The JTRS Step 1 ADR described the over-the-air networking components as existing as applications with IDL interfaces and CORBA middleware as the transfer mechanism between them. Concerns over performance and commercial acceptability of such an architecture were expressed in the JTRS Networking Working Group. The JTRS Networking Working group undertook a system engineering analysis effort to determine whether the networking architecture, described in the ADR, was a valid one or whether some other architecture would be more appropriate. The analysis and subsequent scoring of the transfer mechanism options against criteria derived from ORD requirements essentially reaffirmed the architecture defined in the ADR. Portability was the differentiating factor in the analysis and therefore CORBA was established as the standard transfer mechanism for Networking APIs. The Architecture IPT later adopted the Networking Working Group's approach to services for their APIs for all SCAS services. The requirement for CORBA as the standard transfer mechanism was then generalized to include all services their APIs.

4.2.2.2 Alternate Transfer Mechanism

When an alternate transfer mechanism is used for real-time control and data flow, the transfer mechanism for initialization and non-real-time control shall use the standard transfer mechanism (if those controls can be separated).

The initialization and non real-time control and status in most cases, will be sent/received by an HCI either directly or indirectly through the CF. To retain location transparency between the controlled and controlling entities the interface to the controlled entity must use CORBA as the transfer mechanism.

When an alternate transfer mechanism is used, the transfer and message syntax of the alternate transfer mechanism shall be mapped to the IDL of the API Service Definition.

Mapping the transfer and message syntax of the alternate transfer mechanism to an existing API service definition allows reuse of large portions of the existing definition (behavior, state, etc). It also provides a translation from the alternate transfer mechanism interface to the IDL of the API Service Definition for the purpose of creating an adapter.

This mapping shall be identified by a UUID (separate from the Service Definition UUID).

Just as with the service definition, a mapping of an alternate transfer mechanism to a service definition is required. The requirements for interoperability and future plug-and-play require that this mapping be uniquely and unambiguously identified to ensure that different software objects communicate using an agreed-upon interface definition. Portability requires interfaces (whether publicly or privately documented) to be uniquely and unambiguously identified to ensure that a software object ported from one SCAS Radio System to another communicates with a software object that supports the desired interface rather than another object that supports an identically identified but different interface. For further information, refer to section 4.3

The mapping of an alternate mechanism to the Service Definition, and the UUID that identifies this mapping, aid in tracking and management of the implemented interfaces. It supports interoperability and compatibility.

The description of the alternate transfer mechanism, an analysis supporting the performance need for the alternate mechanism, the mappings to the Service Definition, and the associated UUIDs shall be registered as defined in section 4.3.3.

To maintain a high degree of interoperability and portability with the transfer mechanism implemented in different JTRS domains, it is required that CORBA be used. It is understood that there will likely be wideband waveforms that will have throughput requirements that may not be supportable by a CORBA transport. To support the cases where it can be shown through analysis that the CORBA transport would not meet the data throughput, alternate transfer mechanisms are allowed. The requirement to provide analysis forces the waveform implementers to analyze the requirements and the capabilities. The requirement to publish this information provides other waveform implementers with insight into cases where the standard mechanism was not adequate and gives the Government understanding and justification for the deviation from the standard transfer mechanism.

4.2.2.2.1 API Instance Behavior

Irrespective of the transfer mechanism used, all behavior including state transitions and priorities defined in the service definition shall be obeyed by an API Instance.

Without requiring that an API instance strictly conform to service definition there is no way to guarantee compatibility and interoperability at the interface. This requirement seeks to insure that the required components of the interface defined by the developer are obeyed to provide for compatibility and interoperability.

4.2.2.2.2 Alternate Transfer Mechanism Standards

Transfer mechanisms shall be in accordance with commercial or government standards.

It is in the best interest of the JTRS community to define, adopt, and foster the use of commercial and government standards when developing radio components. Conformance to an open standard is an important criterion for any proposed transfer method that should not be waived without strong technical rationale. The use of standard transfer mechanism(s) also results in lower acquisition and maintenance costs because the standard has already been developed and used thus benefiting from any economies of scale that have taken effect.

4.2.2.2.3 Alternate Transfer Mechanism Selection

There are no requirements levied in this section.

This section does describe goals of, considering the availability of supporting products (that have wide usage), are available from multiple venders, and are expected to have long-term support in the industry. Transfer mechanisms possessing these attributes can have a lower overall cost of ownership since they tend to have lower acquisition and maintenance costs.

4.3 SERVICE DEFINITION REQUIREMENTS

4.3.1 Non-JTRS Service Definitions

Non-JTRS APIs that do not have IDL interfaces shall have a mapping to an IDL interface in a Service Definition as shown in figure 4-1 (4-2 in the API Supplement)**.**

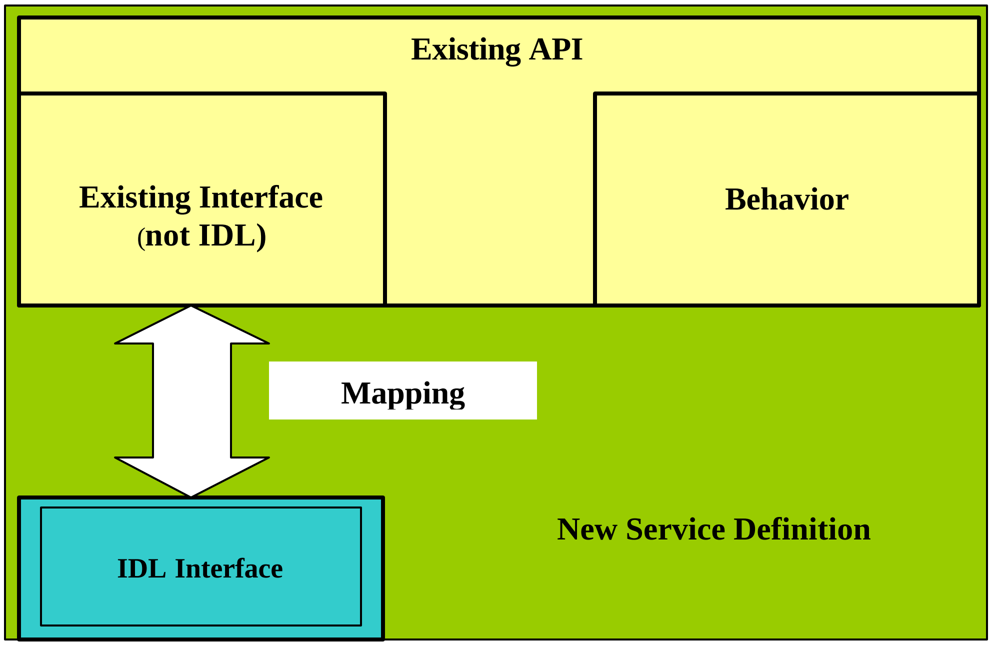

Figure 4-1. Reusing an Existing API Without an IDL Interface

This requirement allows the use of standards with commercial and/or government acceptance. Needless creation of new documentation should be avoided when all that may be required is a mapping of IDL to an existing service definition.

4.3.2 JTRS API Service Definition Format

SCA-compliant APIs shall be defined in Service Definitions conforming to the Service Definition Description (SDD) provided in Appendix A, except as allowed in 4.3.1.

SCA-compliant Service Definitions shall conform to the Service Definition Description (SDD) provided in Appendix A, except as allowed in 4.3.1.

The SDD was developed to provide a single standardized language that could be used when publishing service definitions. A single standardized language will provide for consistency that will allow implementers to quickly access in formation from one SD to another. The information required in the SDD was modeled after commercial standards and extracts the needed information about the interface for easy access by implementers. There is no requirement that a Service Definition be limited to the information included in the SDD. It must include that portion defined in the SDD as a minimum.

4.3.3 Service Definition Identification and Registration

Service Definition documentation of SCA-compliant APIs shall be submitted to a Registration Body to be established, initially, by the JTRS JPO.

Registration of a service definition, with a controlling body will allow for a single point for the collection and distribution of this information. Registered service definitions will provide an “official” standard for any waveform implementers interested in adopting a published service definition. In addition, to be uniquely and unambiguously identified, Service API definitions must be made available if waveform implementers are to develop software that is interoperable and portable. In particular, any Service API definitions which are to be considered "public" must be made available to all vendors. A registration body provides a convenient and single location for vendors to register their public interface definitions and associated UUIDs. Other waveform implementers can then go to the registration body repository and re-use previously developed interfaces, thus increasing potential interoperability and plug-and-play and minimizes the effort required to develop new interfaces.

The registration body could encourage standardization and interoperability by categorizing the interfaces published for different waveforms as mandatory, recommended, allowable, experimental, etc. For example, four interfaces could be publicly registered for a SINCGARS-waveform Modem Networking API. The registration body could declare that one of these interfaces (say the simplest) is mandatory, another (say one that adds extensions to reduce co-site interference) is recommended, and the other two are allowable or experimental. SINCGARS-waveform implementers now know that supporting the mandatory interface will ensure that their software is compatible with other software SINCGARS-waveforms. In addition, if the implementers want to expend extra effort for improved performance, they can also implement the interface that supports improved co-site interference mitigation. Vendors are still free, however, to extend existing interfaces (or even develop new interfaces) to differentiate their product offerings from other vendors.

If the registration body categorizes the published interfaces, then it should be an open standards body similar to the IEEE, SDRF, IETF, etc. in order to reduce any unfair competitive advantage/bias that a single company or private consortium would have. Note that multiple registration bodies may be established. For example, there may be one registration body for military unique waveforms, one registration body for PCS waveforms, and one registration body for wireless LAN waveforms.

Each Service Definition shall be identified by a Universally Unique Identifier (UUID). As used in this specification, the UUID is defined by the DCE UUID standard (adopted by CORBA). No centralized authority is required to administer UUIDs (beyond the one that allocates IEEE 802.1node identifier MAC address).

The association of a Universal Unique ID (UUID) to each service definition will aid in the tracking and management of the implemented interfaces. The UUID can also be used by the Domain Manager to identify which APIs are supported by a particular JTRS resource.

The requirements for interoperability and future plug-and-play require service definition interfaces to be uniquely and unambiguously identified to ensure that different software objects communicate using an agreed-upon interface definition. The Domain Manager can use such an interface identifier as one of the factors in determining which software objects to instantiate together to form a protocol stack. (For example, if an IP network protocol could identify which Link Networking API Service Definition(s) it supported, the Domain Manager could use this identifier to consider only those LinkResources that supported the identified Link Networking API Service Definition.) Thus, there is a requirement for a unique and unambiguous way to identify Service APIs.

The Universally Unique Identifier (UUID) used in CORBA (similar to the Globally Unique Identifier used in COM/DCOM) provides a mechanism to uniquely and unambiguously identify system elements (such as objects and interfaces). The UUID is a 128-bit (16-byte) number that is algorithmically created using a machine's unique network card ID, the current time, and other data. No centralized authority is required to administer UUIDs (beyond the one that allocates IEEE 802.1 node identifiers [MAC addresses]). Because the UUID was mandated as standards based, CORBA compatible way to support the requirement for unique and unambiguous identification of Service APIs.